INSTITUTO FEDERAL DE SANTA CATARINA

YAN LUCAS MARTINS

Protocolo com Múltiplos Canais Assíncronos sobre o protocolo USB-CDC

São José - SC

junho/2023

PROTOCOLO COM MÚLTIPLOS CANAIS ASSÍNCRONOS SOBRE O PROTOCOLO USB-CDC

Monografia apresentada ao Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Prof. Marcelo Maia Sobral, Dr.

Coorientador: Prof. Emerson Ribeiro de Mello, Dr.

São José - SC

junho/2023

Yan Lucas Martins

Protocolo com Múltiplos Canais Assíncronos sobre o protocolo USB-CDC

Este trabalho foi julgado adequado para obtenção do título de Engenheiro de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 07 de julho de 2023:

Prof. Marcelo Maia Sobral, Dr.
Orientador
Instituto Federal de Santa Catarina

Prof. Eraldo Silveira e Silva, Dr.
Instituto Federal de Santa Catarina

Prof. Roberto de Matos, Dr.
Instituto Federal de Santa Catarina

Dedico este trabalho a minha família.

AGRADECIMENTOS

Agradeço a meus pais que nunca mediram esforços para que eu e meu irmão Ygor pudéssemos ter nossos estudos como prioridade, pelo constante apoio, compreensão e incentivo que me deram durante toda a minha trajetória acadêmica. Suas palavras de encorajamento foram fundamentais para superar os desafios e obstáculos encontrados.

Agradeço a minha tia Silvia, cuja sabedoria e experiência foram uma fonte inestimável de inspiração ao longo deste processo.

Agradeço ao meu orientador, professor Sobral, pela atenção e paciência durante o desenvolvimento deste projeto, sendo um grande motivador para a finalização do trabalho.

Agradeço ao meu coorientador, professor Emerson, e ao professor Roberto de Matos, por me trazerem o tema deste trabalho e me auxiliar a formalizar uma proposta e orientação que me ajudaram no decorrer do trabalho.

Agradeço ao Instituto Federal de Santa Catarina - Câmpus São José e a todos os seus colaboradores pelo excelente trabalho realizado e pelo compromisso em fornecer uma educação pública de qualidade.

Por fim, expresso minha gratidão aos meus amigos e a todos que fizeram parte dessa jornada de alguma forma. Sua presença, apoio e encorajamento foram essenciais. Agradeço por compartilharem comigo momentos de estudo, celebrações e desafios, tornando essa jornada acadêmica mais enriquecedora e prazerosa. Muito obrigado a todos!

"Se te serve de algo, nunca é tarde ou, no meu caso, cedo demais para ser quem você quer ser. Não há limite de tempo. Comece quando quiser. Você pode mudar ou não. Não há regras nesse sentido. De tudo podemos obter uma leitura positiva ou negativa. Espero que você obtenha o positivo. Espero que veja coisas que o surpreendam. Espero que sinta coisas que nunca sentiu. Espero que você conheça pessoas com outro ponto de vista. Espero que você viva uma vida da qual se orgulhe. E se você perceber que não é assim, espero que tenha forças para começar de novo."

Benjamin Button

RESUMO

A tecnologia *Universal Bus Serial* (USB) tem sido amplamente utilizada para conectar dispositivos periféricos a computadores e outros dispositivos eletrônicos. No entanto, o desenvolvimento de *drivers* personalizados para cada dispositivo USB pode ser um processo demorado. Bem como, a compatibilidade entre diferentes sistemas operacionais e dispositivos pode ser um desafio significativo, o que acarreta que sistemas operacionais e plataformas menos populares não recebam suporte ao periférico. Outro ponto, é que a comunicação USB ocorre sobre um único canal de comunicação, sendo limitante em cenários onde se necessitem um tratamento de dados que sejam originados em diferentes fontes. Diante disso, neste trabalho, é proposto um protocolo USB com múltiplos canais assíncronos que permite que dispositivos se comuniquem com um *host* utilizando o modelo metres-trabalhador. O protocolo é projetado para ser independente do dispositivo e do sistema operacional, eliminando a necessidade de *drivers* específicos. Será utilizada a classe *Communication Device Class* (CDC) do protocolo USB devido a sua compatibilidade com os maiores sistemas operacionais em uso e por seus modos de alta transferência de dados. O protocolo será responsável por prestar serviços multiplexação de dados, controle de prioridades, estabelecimento de conexão, negociação de parâmetros de comunicação e delimitação de quadro, os quais são essenciais para o funcionamento da comunicação.

Palavras-chave: Protocolo USB. Canais assíncronos. Múltiplos canais.

ABSTRACT

USB technology has been widely used to connect peripheral devices to computers and other electronic devices. However, developing custom drivers for each USB device can be a time consuming process. As well as, compatibility between different operating systems and devices can be a significant challenge, resulting in less popular operating systems and platforms not receiving peripheral support. Another point, is that the USB communication occurs over a single communication channel, being limiting in scenarios where you need a data treatment that originate in different sources. Given this, in this work, a protocol USB with multiple asynchronous channels is proposed that allows devices to communicate with a *host* using the master-worker model. The protocol is designed to be device and operating system independent, eliminating the need for specific drivers. The CDC class of the USB protocol will be used due to its compatibility with major operating systems in use and for its high data transfer modes. The protocol will be responsible for providing services data multiplexing, priority control, connection establishment, communication parameter negotiation and frame delimitation, which are essential for the communication to work.

Keywords: USB protocol. Asynchronous channels. Multiple channels.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ASCII *American Standard Code for Information Interchange.*

CDC *Communication Device Class.*

CRC *Cyclic Redundancy Check.*

ETX *end of text.*

HID *Human Interface Device.*

ISO *International Standards Organization.*

MAC *Media Access Control.*

OTG On-The-Go.

PMCA Protocolo de Múltiplos Canais Assíncronos.

PROMELA *Protocol Meta Language.*

SPIN *Simple Promela Interpreter.*

STX *start of text.*

USB *Universal Bus Serial.*

SUMÁRIO

1 INTRODUÇÃO

Dispositivos USB estão presentes no dia a dia de uma grande parcela da população. Dentre eles pode-se destacar *mouses*, teclados, impressoras, *modens*, telefones e uma infinidade de outros dispositivos. E apesar da sua simplicidade de uso, quando conecta-se um dispositivo USB a um *host*, sendo o último geralmente um computador, pode haver a necessidade da instalação de *drivers*, os quais são responsáveis por garantir a troca de informação entre o dispositivo USB e o *software* do *host* (COMPAQ et al., 2000). No entanto, para cada sistema operacional diferente que um *host* possa ter, por vezes é necessário desenvolver um *driver* específico para o mesmo, visto que os *drivers* de dispositivo USB são implementados com base em um determinado sistema operacional e plataforma (KARNE et al., 2012).

A especificidade ao desenvolver um *driver* para cada tipo de *host*, pode implicar em um prazo maior para o desenvolvimento de um dispositivo USB, pois cada sistema operacional possui suas particularidades de funcionamento sendo necessário conhecer individualmente como cada um opera, o que se mal implementado, impede que a comunicação entre dispositivo-*host* ocorra. Além disso, a necessidade de desenvolver uma massiva quantidade de *drivers* implica em um custo elevado de produção do periférico. Bem como, origina-se a necessidade de escolha, impedindo que um dispositivo USB seja compatível com qualquer *host*, acarretando que apenas sistemas operacionais mais populares possam receber a compatibilidade com o periférico. A partir desta dificuldade, este trabalho propõe um protocolo que irá operar sobre a classe USB-CDC, a qual é capaz de operar em modos de alta transferência de dados. Desta forma, o único requisito que o dispositivo USB e o *host* devem possuir é o de prover suporte ao *driver* dessa classe.

O protocolo deverá suportar uma comunicação assíncrona. Neste modo de transmissão, ao enviar uma mensagem, o dispositivo permanece em um estado não bloqueante, permitindo que o mesmo possa realizar outra atividade independente de ter recebido ou não uma resposta do *host*. O *host*, por sua vez, irá responder apenas quando quiser ou quando puder. A qual se difere da síncrona, que é quando o dispositivo envia uma mensagem, e esse permanece em um estado bloqueado, ou seja, não é capaz de realizar qualquer outra atividade a não ser aguardar a resposta do *host*. Neste cenário, somente uma mensagem por vez é trafegada pelo canal de comunicação e em um único sentido.

O protocolo proposto também deverá prover a comunicação com um ou múltiplos canais, permitindo que mais de uma atividade possa ser executada pelo dispositivo USB e *host*. Desta forma, o protocolo deve ser capaz de garantir o gerenciamento das mensagens vindas de diferentes fontes. Tal garantia pode ser determinada através do estabelecimento

de sessões para cada canal ativo e mecanismos de controle de acesso ao meio, sendo esse meio o barramento USB. O acesso ao meio pode ser dado de três formas: por meio de um sistema de particionamento do canal por tempo, particionamento aleatório ou revezamento entre canais.

Outro obstáculo é a limitação de banda existente em um barramento USB, o qual, também, está suscetível à versão do dispositivo USB que o ocupa (COMPAQ et al., 2000). Logo, é necessário conhecer esses limites para que o empacotamento de mensagens esteja de acordo com o requisito do periférico.

Considerando as dificuldades de desenvolver *drivers* exclusivos a cada sistema operacional e o heterogêneo mercado de dispositivos compatíveis com o protocolo USB, neste trabalho é proposto uma implementação de um protocolo denominado Protocolo de Múltiplos Canais Assíncronos (PMCA), que visa compatibilizar dispositivos USB com *hosts* fazendo uso de um único *driver* independente do sistema operacional utilizado.

1.1 Objetivo geral

O objetivo geral deste trabalho é especificar e desenvolver um protocolo múltiplos canais assíncronos sobre o protocolo USB-CDC, que seja capaz de atuar como uma camada de abstração sobre o protocolo USB. O protocolo desenvolvido, deve evitar a necessidade de instalar *drivers* adicionais no sistema operacional do *host*, empacotar e desempacotar mensagens e comunicar-se em modo de múltiplos canais assíncronos. Deste modo, atuará como um *middleware* que busca minimizar o tempo de desenvolvimento de dispositivos USB.

1.2 Objetivos específicos

Para atingir o objetivo geral, foram definidos os seguintes objetivos específicos:

1. Identificar os serviços essenciais que o protocolo deve atender;

2. Definir um procedimento que garanta o envio de mensagens em um meio concorrido com múltiplos canais assíncronos;

3. Desenvolver uma prova de conceito da solução;

4. Elaborar um plano de testes para a validação da proposta.

1.3 Organização do texto

O presente documento está organizado da seguinte forma: o Capítulo 2 aborda os principais fundamentos teóricos julgados necessários para o desenvolvimento da compreensão e execução deste trabalho, introduzindo o funcionamento do USB, princípios de projeto e propriedades desejáveis de protocolos. Já o Capítulo 3, apresenta a proposta de desenvolvimento do trabalho. Por sua vez, o Capítulo 4 elenca os cenários de testes utilizados para a validação da proposta e seus resultados. E por fim, no Capítulo 5 tem-se a documentação da conclusão deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 USB

O USB é um padrão que define cabos, conectores e protocolos usados em um barramento para conectar, comunicar e fornecer energia entre computadores e outros dispositivos eletrônicos. Computadores pessoais, *tablets* e telefones, que têm portas USB, podem se conectar a dispositivos, desde teclados, *mouses* e *joysticks* a câmeras, impressoras, dispositivos de áudio, vídeo e outros. O USB é uma tecnologia versátil, confiável, rápida, de baixo consumo energético, barata e de amplo uso que provê suporte a diversos sistemas operacionais (AXELSON, 2015).

2.1.1 Arquitetura USB

Para que haja comunicação USB, sua arquitetura requer no mínimo dois elementos: um *host* USB e dispositivos USB. O *host* pode ser um computador pessoal, um dispositivo portátil ou qualquer outro sistema que contém um *hardware* de *host controller* USB e um *root hub*. O *host controller* possui todas as permissões sobre o barramento. Sua responsabilidade, juntamente com o *root hub*, é detectar a conexão e remoção de periféricos USB, gerenciamento do fluxo de controle e dados entre *host* e os dispositivos USB, bem como o fornecimento de energia para o periférico. Toda comunicação em um barramento são iniciadas pelo *host*, deste modo, não pode haver comunicação direta entre dispositivos USB (AXELSON, 2015).

Por sua vez, os dispositivos USB, de maneira sucinta, são todos aqueles que compreendem e respondem operações do protocolo USB e que não possuam um *hardware* de *host controller* USB, estando esses divididos em duas categorias, *hub* e função. O conjunto dessas duas categorias é denominado dispositivo composto (COMPAQ et al., 2000).

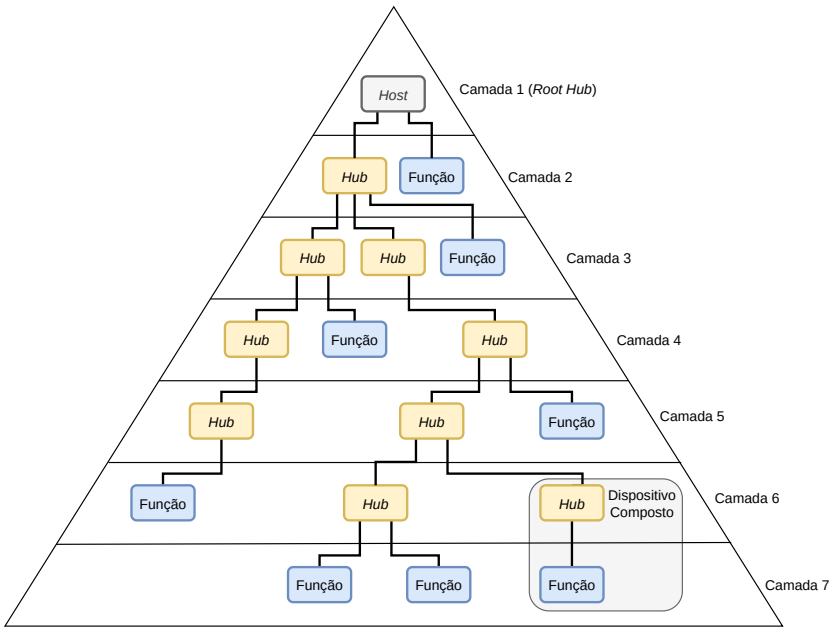
- ***Hub***: cada *hub* converte um único ponto de conexão em vários pontos de conexão chamados de portas USB.

- **Função**: fornece recursos ao *host*, exemplos de um dispositivo deste tipo são: *mouse*, teclado ou impressora. Cada dispositivo de função contém informações de configuração que descrevem os recursos do mesmo, a fim de serem reconhecidos pelo *host*.

- **Dispositivo Composto**: um dispositivo composto contém um *hub* USB e um ou mais dispositivos de função.

A rede USB física é baseada em uma topologia estrela em camadas, na qual há um dispositivo principal, o *host*, e até 127 dispositivos trabalhadores, o que inclui dispositivos de função e *hubs*. Sendo esses divididos em sete camadas: a primeira para o *host*, a sétima para dispositivos de função e as cinco demais camadas para *hubs* e dispositivos de função. Um dispositivo de função pode ser conectado a um *hub*, e esse *hub* pode ser conectado a outro *hub* e assim sucessivamente; desde que respeite o limite de sete camadas, conforme Figura 1 (COMPAQ et al., 2000).

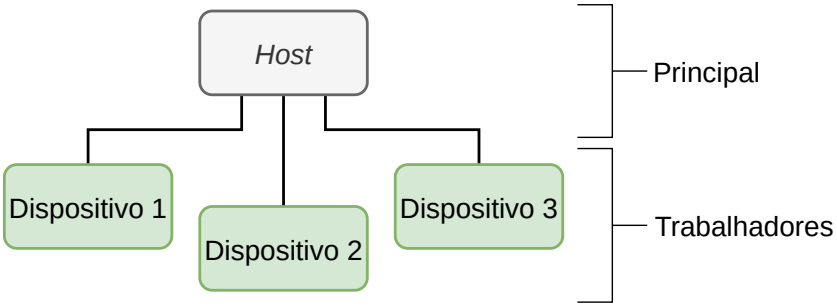
Figura 1 – Topologia de uma rede física de dispositivos USB.



Fonte: Adaptada de (Compaq et al. (2000))

A rede USB lógica, por sua vez, aparece apenas como uma rede em estrela, sendo o centro da mesma, o *host*, conforme exibe a Figura 2. Os *hubs* são transparentes nesta perspectiva, sendo assim, o dispositivo USB deve funcionar da mesma maneira se conectado diretamente ao *host* ou por meio de *hubs* intermediários (AXELSON, 2015).

Figura 2 – Topologia de uma rede lógica de dispositivos USB.



Fonte: Adaptada de (Compaq et al. (2000))

Todos os dispositivos USB estão disponíveis com nós endereçáveis nesta rede mestre-trabalhador, graças a um endereço único fornecido pelo *host* para cada dispo-

sitivo USB presente na rede. Quando o *host* está transmitindo um pacote de dados, ele é enviado a todos os dispositivos conectados. Cada *hub* ressincroniza as transições de dados à medida que os retransmite. Unicamente um dispositivo, o endereçado, realmente aceita os dados; todos os outros recebem, mas os rejeitam. Apenas um dispositivo por vez é capaz de transmitir para o *host*, em resposta a uma solicitação direta dele. Cada *hub* repete todos os dados que recebe de uma camada inferior apenas em direção da camada superior (AXELSON, 2015).

2.1.2 Transferência de dados

Os dispositivos USB são classificados por velocidade do barramento. A versão 2.0 do USB (COMPAQ et al., 2000) especifica três dessas velocidades: *Low Speed*, *Full Speed* e *High Speed* conforme Tabela 1. Na versão 3.2 (APPLE et al., 2017), ainda são especificadas mais duas velocidades para um barramento: a *Super Speed* e a *Super Speed Plus*.

Tabela 1 – Vazão USB por especificação.

Velocidade	**Vazão máxima teórica**	**Aplicação**
Low Speed	1,5 Mbps	Dispositivos interativos: *mouse*, teclado e periféricos para jogos
Full Speed	12 Mbps	Áudio, impressoras e *scanners*
High Speed	480 Mbps	Vídeo e armazenamento em massa
Super Speed	5 Gbps	Vídeos em alta definição e armazenamento em massa
Super Speed Plus	10 Gbps	Vídeos em alta definição e armazenamento em massa

Fonte: Adaptada de Microchip (2020)

As transferências de dados USB ocorrem por meio de uma série de eventos chamados transações. Os pacotes de uma transação são classificados em: *token*, o qual descreve o tipo da transação; *data*, sendo este a carga útil de uma transação; e *handshake*, o qual define o estado de uma solicitação de comunicação. As transações são conduzidas em um intervalo de tempo controlado pelo *host*, denominado *frame*. A duração e a frequência das transações dependem do tipo de transferência que está sendo usado por um *endpoint*, conforme detalhado na subseção 2.1.3. O USB *frame* dura 1 milissegundo para dispositivos configurados em *Full* e *Low Speed* e 125 microssegundos para barramentos com *High Speed* (COMPAQ et al., 2000).

2.1.3 Modos de transferência USB

Os *endpoints* ou pontos de extremidade, podem ser descritos como fontes ou coletores de dados em um dispositivo USB, estando localizados na extremidade do fluxo de comunicação entre um dispositivo USB de função e um *host*. Devido a restrição de ocupação do barramento, os *endpoints* devem aguardar uma ordem de envio ou recepção de

dados, vindo diretamente do *host*. Quando um dispositivo de função possui dados a serem enviados, cabe aos *endpoints* manter esses dados armazenados em *buffer*, aguardando até que recebam a autorização para a transmissão dos mesmos pelo barramento (COMPAQ et al., 2000).

É previsto na versão 2.0 do protocolo USB (COMPAQ et al., 2000) quatro tipos de *endpoint descriptors*, isto é, modos de transmissão diferentes para cada *endpoint* que um dispositivo USB possa ter. Cada modo fornece características diferentes de tratamento de erros, latência garantida, largura de banda e sentido de transmissão (COMPAQ et al., 2000). Esses modos de transmissão são introduzidos nos itens a seguir:

- ***Control transfer***: este modo é responsável por comunicações de comando, *status* ou configuração entre o dispositivo e o *host*. Todo dispositivo USB deve possuir pelo menos um *endpoint* com este modo de transferência, denominado de *endpoint 0*. Normalmente a transmissão dos pacotes ocorre em rajadas, iniciada pelo *host* e sem garantia de recebimento.

- ***Isochronous transfer***: neste modo existe uma garantia de largura de banda, taxa de transmissão e latência limitada. O sentido da transmissão será sempre unidirecional. Existe a detecção de erros porém sem retransmissão. A característica da comunicação é contínua e periódica e geralmente contém informações sensíveis ao tempo como *stream* de áudio e vídeo.

- ***Interrupt transfer***: este modo de transmissão é utilizado quando o dispositivo precisa enviar ou receber informação do *host* sem uma determinada frequência porém com um período de transmissão bem determinado. Desta forma, este modo de operação garante um máximo período de transmissão para o dispositivo, bem como tentativas de retransmissão no próximo período em caso de erros de transmissão.

- ***Bulk transfer***: este modo é utilizado quando o dispositivo precisa transmitir grandes quantidades de informação e utilizar o máximo de largura de banda disponível no momento. Desta forma, este modo possui retransmissão em caso de erro, garantia de entrega dos dados transmitidos. No entanto, não é garantido neste modo largura de banda ou baixa latência.

Cada modo de transferência possui um limite de carga útil diferente, por consequência, emprega diferentes taxas de transferência para cada velocidade do barramento. Para melhor observação, esses dados foram estruturados na tabela Tabela 2.

O endereço de *endpoint* consiste em um número do *endpoint* e uma direção. O número do *endpoint*, para a transferência de dados, é um valor inteiro no intervalo de 1 a 15. A direção, por sua vez, é definida a partir da perspectiva do *host*: um *endpoint* IN

Tabela 2 – Características de vazão no barramento por modo de transmissão

Velocidade	**Modo de transmissão**	**Carga útil máxima**	**Transferência por frame**	**Vazão máxima teórica**
Low Speed	Control	8 bytes	1	8 kB/s
Low Speed	Interrupt	8 bytes	1	8 kB/s
Low Speed	Bulk	não se aplica	não se aplica	não se aplica
Low Speed	Isochronous	não se aplica	não se aplica	não se aplica
Full Speed	Control	64 bytes	1	64 kB/s
Full Speed	Interrupt	64 bytes	1	64 kB/s
Full Speed	Bulk	64 bytes	até 19	1.2 MB/s
Full Speed	Isochronous	1023 bytes	1	1023 kB/s
High Speed	Control	64 bytes	1	512 kB/s
High Speed	Interrupt	1024 bytes	até 3	24 MB/s
High Speed	Bulk	512 bytes	até 13	53 MB/s
High Speed	Isochronous	1024 bytes	até 3	24 MB/s

Fonte: Microchip (2020)

fornece dados para enviar ao *host* e um *endpoint* OUT armazena os dados recebidos do *host*. O par zero é reservado ao *endpoint* de controle denominado *endpoint* zero. Este é o ponto que recebe todas as solicitações de controle e *status* dos dispositivos durante todo o tempo em que o dispositivo está operacional no barramento (COMPAQ et al., 2000).

2.1.4 Descritores USB

Dispositivos USB transmitem suas informações por meio de descritores, que são blocos de dados com formatos definidos e estruturados de forma hierárquica. Cada dispositivo USB possui quatro descritores padrão (COMPAQ et al., 2000):

- ***Device Descriptor***: representa as informações essenciais do dispositivo USB, existindo apenas um descritor deste tipo por periférico. O descritor possui informações como versão do protocolo, identificação de fabricante e a quantidade de *configuration descriptors*, por exemplo;
- ***Configuration Descriptor***: um dispositivo USB pode ter um ou mais *configuration descriptors*. Este é responsável por indicar informações como número de interfaces, consumo de energia do dispositivo e modo de alimentação;
- ***Interface Descriptor***: abaixo do *configuration descriptor* pode haver um ou mais *interface descriptors*. Este, por sua vez, é responsável por indicar a classe USB utilizada, bem como indicar quantos *endpoint descriptors* estão abaixo desta interface;
- ***Endpoint Descriptor***: este descritor é utilizado pelo *host* para, por exemplo, determinar o total de largura de banda que deve ser alocado pelo barramento, além

de definir o modo de operação do *endpoint*, o que impacta diretamente na taxa de transferência do dispositivo.

2.1.5 Classes USB

A especificação do protocolo USB (COMPAQ et al., 2000) define classes para diferenciar e identificar as funcionalidades de um dispositivo USB, permitindo ao *host* carregar o *driver* necessário de acordo com a classe do dispositivo. Essa divisão, permite que dispositivos com funcionalidades semelhantes possam fazer uso do mesmo *driver*, não havendo a necessidade do desenvolvimento de um *driver* exclusivo para todo dispositivo. Esta seção busca determinar uma classe que ofereça uma máxima taxa de transferência e que possua drivers pré-instalados nos sistemas operacionais mais comuns.

2.1.5.1 Classe HID

A classe *Human Interface Device* (HID) inclui dispositivos, como teclados, *joysticks* e *mouses*. Para todos esses dispositivos, o *host* recebe dados que correspondem à entrada humana, como pressionamentos de tecla e movimentos do *mouse*. Além dos descritores padrão, conforme apresentados na subseção 2.1.4, esta classe possui três descritores específicos: *HID Descriptor*, *Report Descriptor* e *Physical Descriptor*. Um dispositivo com a classe HID se comunica com o *host* a partir de *endpoints* no modo de transferência *control* ou *interrupt* (AXELSON, 2015).

2.1.5.2 Classe CDC

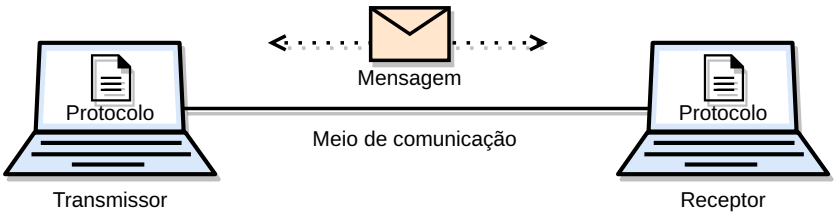
A classe CDC é comumente utilizada em dispositivos de telecomunicações, tal quais *modens* e telefones, além de outras funções de comunicação, como portas seriais virtuais. Em sua especificação é definido que todo dispositivo pertencente a esta classe, deve possuir um descritor de interface denominado *Communications Interface Class*, o qual lida com o gerenciamento de controle do dispositivo. Esse gerenciamento gera notificações que podem ser transferidas utilizando o modo *interrupt* ou *bulk*. Um dispositivo do tipo CDC pode ainda conter uma interface para a troca de dados chamada de *Data Interface Class*. Para a troca de dados pode-se utilizar tanto o modo *bulk* quanto o modo *isochronous* (AXELSON, 2015). Por fazer uso desses dois últimos modos de transferência citados, esta classe permite uma troca de dados maior que a classe HID.

2.2 Protocolo de comunicação

A comunicação de dados pode ser entendida como a troca de informação entre dois dispositivos por meio de algum ambiente de comunicação. Nesse ambiente, temos a mensagem que é a informação a ser transmitida; o transmissor e o receptor, responsáveis

por transmitir e receber a mensagem respectivamente; o meio de comunicação que é o caminho físico por onde a mensagem será enviada; e por fim, o protocolo, o qual define um conjunto de regras para a transmissão e recepção da mensagem. Um modelo simplificado é apresentado na Figura 3.

Figura 3 – Sistema básico de comunicação.



Fonte: Adaptada de (Holzmann (1991))

Todas as regras, formatos e procedimentos acordados entre transmissor e receptor são chamados de protocolo. Essa definição é importante pois formalizando a interação garante-se que os dados serão reconhecidos em ambas as extremidades da comunicação. Nas seções a seguir, são discutidas alguns elementos considerados essenciais em uma definição de protocolo.

2.2.1 Os elementos de um protocolo

Segundo Holzmann (1991), a especificação de um protocolo consiste em cinco elementos. Para uma especificação ser completa deve-se incluir:

1. O **serviço** a ser prestado pelo protocolo;
2. As **considerações** sobre o ambiente de comunicação em que o protocolo é executado;
3. O **vocabulário** das mensagens utilizadas pra a implementação do protocolo;
4. A **codificação** de cada mensagem do vocabulário;
5. O **comportamento** que define as regras para intercâmbio de mensagens.

Holzmann (1991) compara a definição de um protocolo com a definição de um idioma falado: um idioma contém um vocabulário e uma definição de sintaxe, o que representa o formato e codificação do protocolo; as regras de comportamento definem uma gramática; e a especificação do serviço define a semântica da linguagem. Além disso, define que a linguagem do protocolo deve ser inequívoca, estando esta pronta para lidar com situações adversas como o tempo, condições de corrida e possíveis bloqueios.

2.2.2 Serviço e ambiente

Para que uma tarefa de nível superior possa ocorrer, como uma transferência de arquivos, uma variedade de funções de nível inferior, como sincronização e recuperação de erro devem ser executadas. Essas funções devem ser introduzidas no protocolo de acordo com o ambiente que o mesmo irá operar, fazendo-se uma análise do meio de transmissão. A recuperação de erros, por exemplo, deve estar apta a lidar com um meio de transmissão específico, o qual pode ser diverso como um canal ponto a ponto, dedicado à comunicação entre duas máquinas específicas, ou um canal de transmissão compartilhado, como a rede Aloha ou um enlace Ethernet (HOLZMANN, 1991).

O *software* do protocolo, pode ser convenientemente estruturado em camadas. Funções mais abstratas são definidas e implementadas em termos de construções de níveis, onde cada camada oculta uma certa propriedade do canal de comunicação. Um *design* em camadas ajuda a indicar a estrutura lógica do protocolo, separando as tarefas de nível superior dos detalhes de nível inferior. Quando o protocolo deve ser estendido ou alterado, é mais fácil substituir um módulo do que reescrever todo o protocolo (HOLZMANN, 1991).

A *International Standards Organization* (ISO) reconhece as vantagens de padronizar uma hierarquia de serviços de protocolo, e estabelece um modelo de referência para projetistas de protocolo. A qual define sete camadas:

1. **Camada física:** transmissão de *bits* em um circuito físico.
2. **Camada de enlace de dados:** detecção e recuperação de erros.
3. **Camada de rede:** transferência e roteamento de dados.
4. **Camada de transporte:** transferência de dados de usuário para usuário. Trata controle de fluxo, ordenação dos pacotes e a correção de erros.
5. **Camada de sessão:** coordenação de interações em sessões entre *hosts*.
6. **Camada de apresentação:** interpretação da sintaxe em nível de usuário, por exemplo, para criptografia ou compressão de dados. Atua como um tradutor.
7. **Camada de aplicação:** ponto de entrada para processos de aplicação, promove uma interação entre máquina e usuário.

Cada camada na hierarquia define um serviço distinto e implementa um ou mais protocolos diferentes. O formato usado por qualquer camada específica é amplamente independente dos formatos usados pelas outras camadas. A camada de rede, por exemplo, envia pacotes de dados, a camada de enlace de dados os converte em quadros e a camada física os traduz em *bytes* ou fluxos de *bits*. O receptor decodifica os dados brutos na

camada 1, interpreta e exclui a estrutura do quadro na camada 2, para que a camada 3 possa reconhecer novamente a estrutura do pacote. O formato imposto pelas camadas inferiores deve ser transparente para as camadas superiores (HOLZMANN, 1991).

2.2.3 Vocabulário e formato

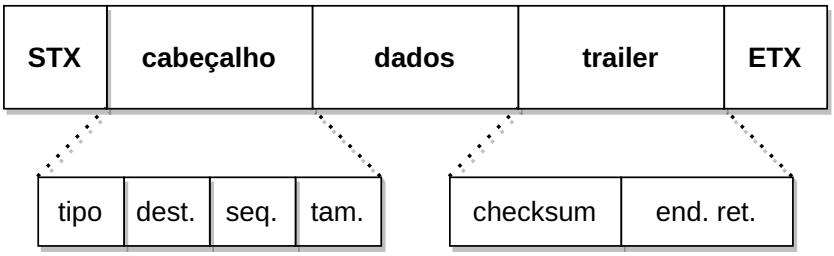
Existem alguns métodos para a formatação de mensagem. De acordo com Holzmann (1991), os três principais métodos de formatação são: orientado a *bits*, orientado a caractere e orientado a contagem de *bytes*. Esses, sem a adição de campos especiais, não previnem e não são capazes de tratar falhas. Em um protocolo orientado a caracteres, por exemplo, há apenas os códigos de controle, que são os bytes *start of text* (STX) e *end of text* (ETX) que servem como delimitadores de uma mensagem.

No entanto, um protocolo está suscetível a erros. Se um campo de contagem de bytes for distorcido ou um caractere de controle for perdido, o receptor poderá não identificar o quadro. São necessários esquemas de detecção de erros, os quais requerem a transmissão de informações redundantes, normalmente na forma de uma soma de verificação (*checksum*). Se forem adicionadas técnicas de controle de fluxo, por exemplo, para detectar perda ou reordenação de quadros de texto, um campo de número de sequência é anexado. Se mais de um tipo de mensagem for usado, deve-se incluir um identificador do tipo de mensagem que está sendo transferida (HOLZMANN, 1991).

Toda essa sobrecarga deve ser agrupada em estruturas separadas que encapsulam os dados. O formato da mensagem pode então ser definido como um conjunto ordenado de três elementos: cabeçalho, dados e trailer, conforme Figura 4. O cabeçalho e o trailer, por sua vez, definem subconjuntos ordenados de campos de controle, que podem ser definidos da seguinte forma:

- **Cabeçalho:** contendo o tipo, destino, número de sequência e tamanho, por exemplo.
- **Trailer:** incluindo, por exemplo, o *checksum* e endereço de retorno.

Figura 4 – Formato da mensagem



Fonte: Adaptada de (Holzmann (1991))

Não há uma regra geral sobre quais campos um protocolo deve possuir, é necessário conhecer o meio e os serviços a fim de julgar os campos vitais para o funcionamento do mesmo. Uma dessas adições é o campo tipo que pode ser usado para identificar as mensagens que compõem o vocabulário do protocolo. Dependendo da estrutura particular do vocabulário do protocolo, este campo pode ser ainda mais refinado (HOLZMANN, 1991).

2.2.4 Propriedades do protocolo

Ainda segundo Holzmann (1991), um protocolo deve possuir algumas propriedades desejáveis: simplicidade, modularidade, adequação, robustez e consistência.

- **Simplicidade:** quando um protocolo pode ser construído a partir de um pequeno número de partes bem projetadas e bem entendidas. Cada parte desempenha uma função e é capaz de executá-la de forma concreta. Para entender o protocolo, deve ser necessário apenas compreender o funcionamento de cada parte e a forma como interagem.

- **Modularidade:** quando cada parte do protocolo possui uma interação simples e bem definida. Cada parte menor é um protocolo que pode ser desenvolvido, verificado, implementado e mantido separadamente das demais partes. A estrutura de protocolo resultante é aberta, extensível e reorganizável, sem afetar o funcionamento das outras partes.

- **Adequação:** não é superespecificado, ou seja, não contém nenhum código inalcançável ou não executável. Tampouco deve estar incompleto. O protocolo deve ser limitado, não ultrapassando os limites do sistema, ser estável e adaptável.

- **Robustez:** deve funcionar em condições normais, bem como em situações imprevistas. Deve ser capaz de lidar com cada possível sequência de ações, em todas as possíveis condições. Deve ter um projeto mínimo, de forma a remover considerações não essenciais que poderiam impedir sua adaptação a condições não previstas.

- **Consistência:** protocolos não devem apresentar interações que os levem a falhar, tais como:

  - ***Deadlocks*:** estados nos quais nenhuma execução de protocolo adicional é possível, ou seja, quando todos os processos de protocolo estão esperando por outras condições que nunca podem ser cumpridas.

  - ***Livelocks*:** sequências de execução que podem ser repetidas indefinidamente, sem nunca realizar um progresso efetivo.

- **Encerramentos inesperados:** a conclusão da execução sem satisfazer as condições de encerramento adequadas.

Em geral, esses critérios não podem ser verificados por uma inspeção manual da especificação do protocolo. Ferramentas mais poderosas são necessárias para evitá-los ou detectá-los. Algumas dessas ferramentas poderiam ser uma linguagem para modelagem de sincronização e coordenação de processos concorrentes como o *Protocol Meta Language* (PROMELA) e o verificador *Simple Promela Interpreter* (SPIN).

3 PROTOCOLO MCA

A proposta deste trabalho é desenvolver um protocolo com múltiplos canais assíncronos que atuará como uma camada de abstração acima do USB-CDC, permitindo a comunicação entre dois dispositivos USB e que o desenvolvedor não tenha a necessidade de criar *drivers* proprietários para interagir com o USB a baixo nível.

A escolha da classe USB foi baseada em dois requisitos principais: compatibilidade com diversos dispositivos e alta taxa de transmissão. Após avaliar as opções disponíveis, optou-se pela classe CDC em detrimento da classe HID devido à sua maior taxa de transmissão. Além disso, a classe CDC possui suporte nativo em sistemas operacionais populares, como Android, Linux, macOS, Windows e outros. Isso elimina a necessidade de desenvolver um driver proprietário para a interface de comunicação entre o *host* e o dispositivo, reduzindo o tempo de desenvolvimento e ampliando o suporte do dispositivo para diferentes sistemas operacionais e plataformas.

O PMCA utiliza a comunicação assíncrona, o que possibilita que os dispositivos USB não fiquem bloqueados enquanto aguardam uma resposta do *host*, permitindo um melhor aproveitamento de seus recursos. Um exemplo desse conceito pode ser observado em aplicativos de mensagens instantâneas, como o WhatsApp. Nesses aplicativos, os usuários podem enviar mensagens sem a necessidade de obter uma resposta imediata, enquanto o destinatário tem a liberdade de responder quando estiver disponível. Essa abordagem oferece flexibilidade na comunicação, permitindo que os dispositivos interajam em seu próprio ritmo.

Este protocolo faz uso do modelo Mestre e Trabalhador, sendo estes o *host* e o dispositivo USB respectivamente. Neste modelo de comunicação, o Mestre assume o papel ativo na troca de mensagens. Ele é responsável por tomar a iniciativa de enviar mensagens ao Trabalhador. O Mestre pode enviar uma variedade de comandos ou solicitações para o Trabalhador. Por outro lado, o Trabalhador assume um papel reativo na comunicação. Ele aguarda a recepção de mensagens enviadas pelo Mestre e responde de acordo. O Trabalhador não toma a iniciativa de enviar mensagens por conta própria, mas sim reage às mensagens que recebe do Mestre. Essas respostas podem incluir informações solicitadas, confirmações de recebimento ou qualquer ação específica que o Trabalhador precise realizar com base nas instruções do Mestre. Essa dinâmica de comunicação entre o Mestre e o Trabalhador permite um fluxo controlado de troca de informações. É um modelo comumente utilizado em sistemas distribuídos e protocolos de comunicação onde há uma entidade principal que coordena e controla as interações com entidades secundárias (CHRISTOFOROU et al., 2014).

A modelagem do protocolo é essencial para assegurar seu funcionamento completo, incluindo todos os serviços que ele irá fornecer baseado no ambiente em que está inserido. Os detalhes sobre esses serviços podem ser encontrados na seção 3.1 deste documento.

3.1 Serviço e ambiente do protocolo

Conforme discutido na subseção 2.2.2, é fundamental conhecer os serviços que o protocolo oferece e em qual meio ele irá operar a fim de determinar a necessidade ou não de possíveis tratamentos de erro e sincronização.

O ambiente deste protocolo é o barramento USB. Logo, a comunicação exercida neste meio, tende a possuir maior segurança e confiabilidade de entrega de dados que em uma rede sem fio, visto que o ambiente não cabeado está mais propenso a ruídos originários de outras transmissões que podem ocorrer no mesmo meio e frequência de transmissão. Em resumo, há casos onde os dispositivos sem fio devem competir com outros dispositivos sem fio, sendo necessário no seu protocolo de comunicação uma camada de *Media Access Control* (MAC). Porém, no barramento USB, este acesso é gerenciado pelo protocolo USB.

O protocolo USB também provê os serviços de garantia de entrega e checagem de erro; caso um pacote chegue corrompido, com *bytes* faltando ou simplesmente não teve uma resposta de que foi entregue, o próprio protocolo USB se encarrega de enviar novamente o pacote de dados até que o mesmo seja entregue ou quando o dispositivo desista de obter uma confirmação de entrega (COMPAQ et al., 2000).

Diante disso, após uma análise da documentação do protocolo USB, as seguintes considerações para o ambiente de funcionamento foram determinadas:

- Presume-se que o canal de transmissão não cause distorções nas mensagens, não duplique, não insira ou reordene as mensagens. Confia-se na robustez do USB, o qual fornece:
  - O sinal será entregue íntegro, visto que o ambiente cabeado possui proteções, como a blindagem do cabeamento;
  - Proteção *Cyclic Redundancy Check* (CRC) no controle e nos campos de dados. O CRC é um teste realizado em um dado para detectar possíveis erros durante a transmissão, leitura e escrita de uma mensagem;
  - Detecção de conexão e remoção de dispositivos;
  - Auto-recuperação do protocolo, usando *timeouts* para pacotes perdidos ou corrompidos.

Tendo em vista o ambiente no qual a proposta de protocolo tema deste documento atuará, foram elencados alguns serviços que foram implementados para esta aplicação:

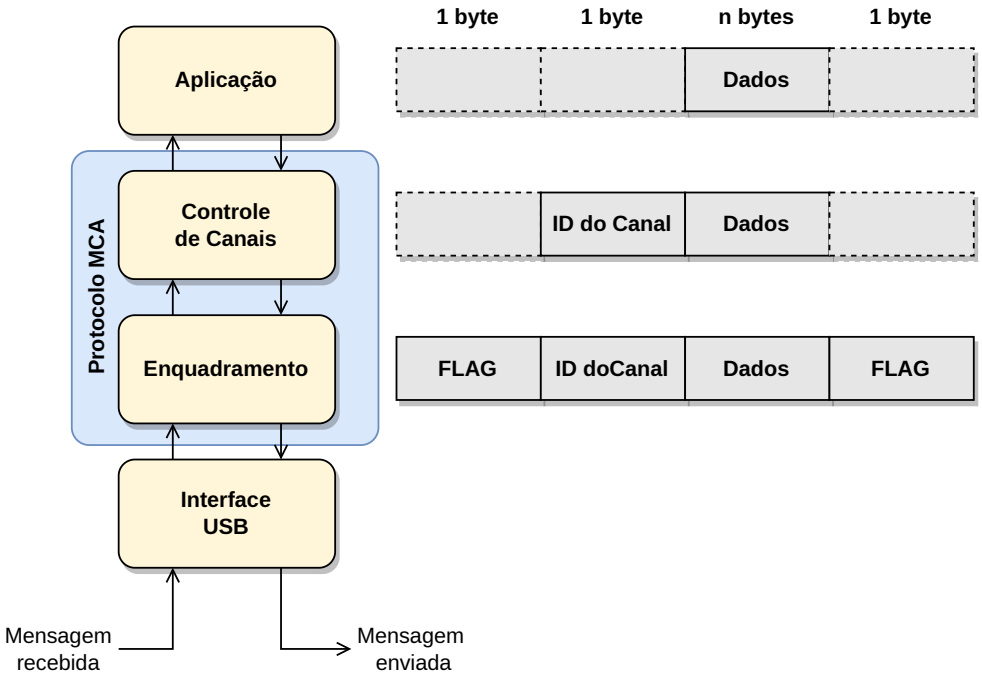
- **Transferência de dados entre dispositivos USB**:
  - **Segmentação de dados:** Os dados podem ou não ser divididos em pacotes de tamanho pré-definido no estabelecimento de conexão. Não há restrições quanto ao tamanho de mensagens enviados por este protocolo, caberá ao usuário que faça uso da API do PMCA determinar essa restrição, se assim desejar.
  - **Multiplexação de Dados:** O protocolo permite que múltiplos fluxos de dados compartilhem o mesmo canal de comunicação.
  - **Controle de Prioridade:** O protocolo permite a atribuição de prioridades aos pacotes de dados para garantir que os pacotes mais críticos sejam transmitidos primeiro. Isso é particularmente útil em aplicações que demandam qualidade de serviço, onde certos dados têm requisitos de latência mais estritos.
- **Estabelecimento de conexão:**
  - **Negociação de parâmetros:** Os dispositivos envolvidos na comunicação, para cada canal criado, negociam os parâmetros das mensagens do protocolo como, limite de canais, tamanho da mensagem e prioridade do canal.
  - ***Handshaking*:** Os dispositivos trocam mensagens de *handshake* para confirmar conexões e desconexões de canais.

Com base nos serviços estabelecidos nesta seção, foi determinado que o protocolo será dividido em duas camadas: Controle de Canais e Enquadramento, as quais estão ilustradas na Figura 5. A aplicação gera os dados que serão enviados pelo protocolo, os quais serão atribuídos a um canal de transmissão pelo módulo de Controle de Canais (seção 3.3). Em seguida, a mensagem será delimitada pelo Enquadramento (seção 3.2) e, por fim, estará pronta para ser enviada por meio da interface USB-CDC.

3.2 Módulo de enquadramento

O módulo de Enquadramento é responsável por delimitar os quadros das mensagens, permitindo que o receptor reconheça quando uma mensagem foi recebida integralmente. Nesta proposta, foi adotado o modelo sentinela como técnica de enquadramento, onde são adicionadas *flags* que determinam o início e fim de cada quadro. Além disso, é necessário utilizar uma *flag* de *escape*, que indica que o *byte* seguinte, cuja denominação neste documento será "*byte* conflitivo", precisa passar por uma conversão para que seu valor original possa ser interpretado corretamente. Neste protocolo, o *byte* de delimitação

Figura 5 – Diagrama de camadas do protocolo.



Fonte: Elaborada pelo autor.

de quadro é o valor, em representação hexadecimal, `7E` e o *byte* de *escape* o valor `7D`. Os *bytes* conflitivos são tratados utilizando uma operação lógica, conforme demonstrado na Equação 3.1, em que x é o valor do *byte* após a conversão e y o valor do *byte* com conflito.

$$x = y \oplus 0x20 \tag{3.1}$$

Para lidar com a situação dos *bytes* conflitivos, a Tabela 3 estabelece o resultado desejado após a conversão desses *bytes*.

Tabela 3 – Tratamento de um *byte* conflitivo e o seu resultado.

Byte conflitivo	Operação	Resultado
`0x7E`	xor `0x20`	`0x5E`
`0x7D`	xor `0x20`	`0x5D`

Fonte: Elaborada pelo autor.

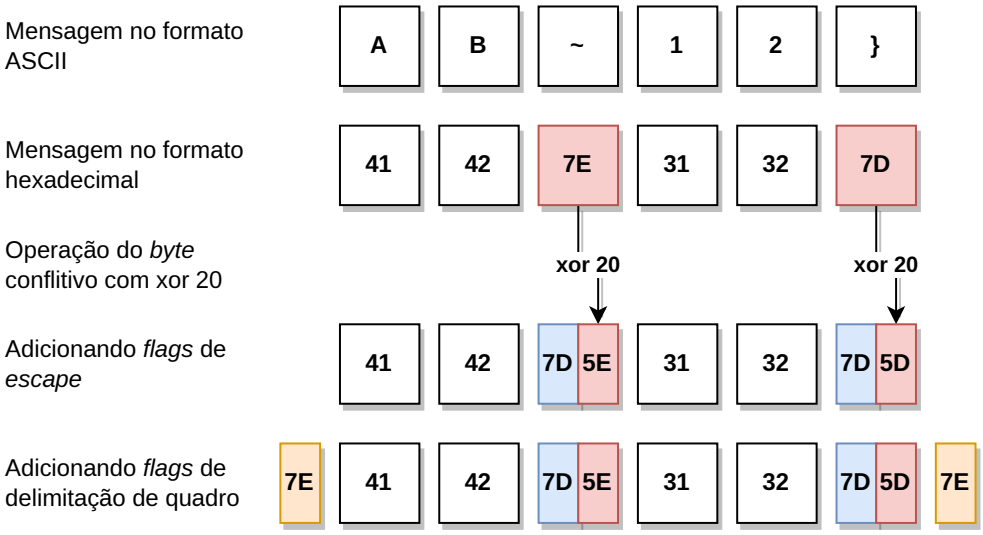
3.2.1 Processo de envio de um quadro

Durante o processo de envio de uma mensagem, é fundamental realizar um tratamento adequado dos *bytes* conflitantes. Caso esse tratamento não seja executado corretamente, a outra extremidade do protocolo terá dificuldades em delimitar corretamente as informações recebidas. Inicialmente, é necessário realizar uma varredura em seus *bytes*,

a fim de identificar possíveis falsas *flags* sentinelas que possam estar presentes no corpo da mensagem. Essas falsas *flags* podem causar problemas na interpretação dos dados recebidos, comprometendo a integridade e a confiabilidade da comunicação. Uma vez que os *bytes* conflitantes são tratados, é necessário adicionar as *flags* de sentinela, que servem para delimitar o início e o término da mensagem. Essas *flags* são o primeiro e pelo último byte do quadro, e sua presença garante uma correta identificação dos limites da informação transmitida. Portanto, ao seguir esse processo de tratamento dos *bytes* conflitantes e adição das *flags* de sentinela, é possível garantir uma transmissão de dados confiável, assegurando que a mensagem seja corretamente interpretada e recebida pela outra extremidade do protocolo.

A Figura 6 ilustra o processo realizado por este módulo para enviar uma mensagem, incluindo a conversão de *bytes* conflitantes e a adição de *flags* delimitadoras e de *escape*. Optou-se por utilizar os formatos *American Standard Code for Information Interchange* (ASCII) e Hexadecimal no exemplo, pois são formatos compreensíveis pelos humanos, o que facilita a visualização.

Figura 6 – Processo realizado pelo módulo de enquadramento para adição dos *bytes* sentinelas.



Fonte: Elaborada pelo autor.

3.2.2 Processo de recepção de um quadro

Na outra extremidade do protocolo, durante a recepção de um dado em outro dispositivo, é necessário implementar um fluxo reverso com algumas adições em relação ao processo anteriormente descrito. O fluxo reverso é composto pelos seguintes estados: `OCIOSO`, `RECEPÇÃO` e `ESCAPE`.

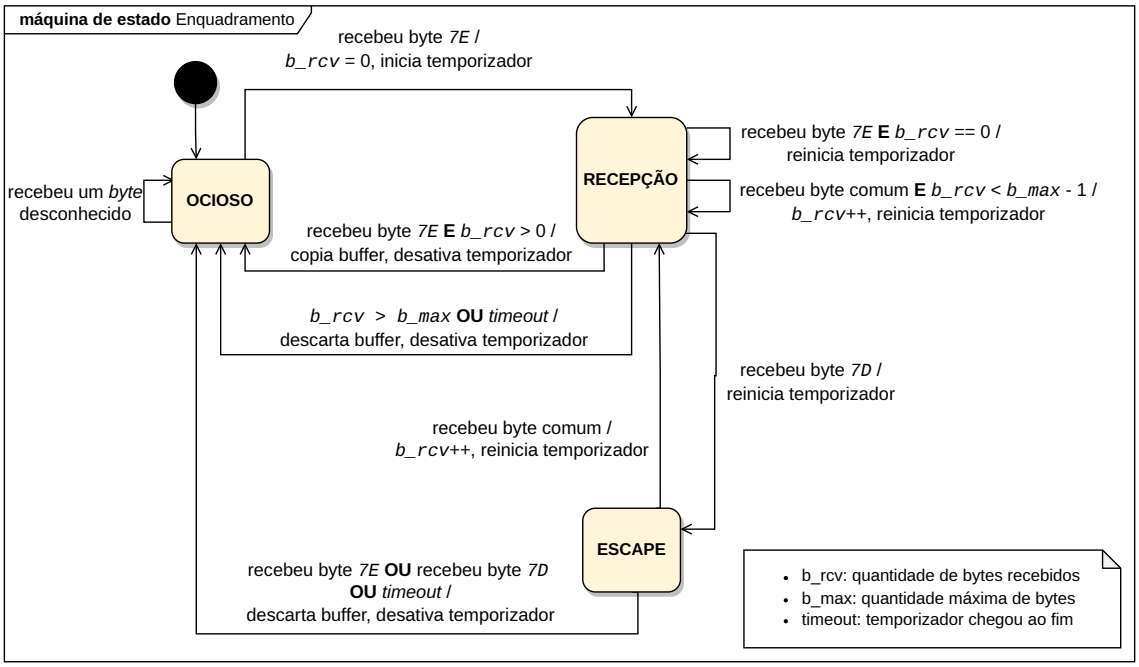
Inicialmente, o enquadramento está no estado `OCIOSO`. Quando um *byte* é recebido, verifica-se se ele corresponde a uma *flag* delimitadora de quadro. Se for o caso, o

enquadramento passa para o estado `RECEPÇÃO`, onde permanecerá até receber um *byte* de finalização de quadro. Esse *byte* fará com que o enquadramento retorne ao estado `OCIOSO`. No entanto, se durante a `RECEPÇÃO` for recebida uma *flag* de *escape*, o enquadramento alterna para o estado `ESCAPE`. Assim que o enquadramento sai do estado `OCIOSO`, é necessário iniciar um temporizador com um tempo de permanência padrão de 5 segundos. Após esse período, se nenhum *byte* for recebido, o enquadramento retorna ao estado `OCIOSO`. No entanto, é relevante destacar que os tempos de permanência podem ser ajustados de acordo com os requisitos específicos do sistema.

Além disso, durante a configuração do protocolo, é possível definir o tamanho máximo do *buffer* de recepção e transmissão. Essa configuração determina a capacidade de armazenamento dos dados que podem ser recebidos ou enviados pelo enquadramento. Dependendo da aplicação, pode ser necessário limitar o tamanho do *buffer* para otimizar o uso de recursos ou garantir a compatibilidade com restrições de memória.

Esse fluxo descrito se repete conforme ilustrado na Figura 7, demonstrando a sequência de estados e transições durante o processo de recepção de dados.

Figura 7 – Máquina de estados finita que representa o comportamento do módulo de Enquadramento.



Fonte: Elaborada pelo autor.

3.3 Módulo de Controle de Canais

Dentre os serviços implementados por este protocolo vistos na seção 3.1, o módulo de controle de canais é responsável pelos serviços de multiplexação de dados, controle de

prioridade, negociação de parâmetros e *handshaking* para o estabelecimento de conexões. De maneira sucinta, este módulo desempenha o papel de determinar o canal responsável por uma transmissão, identificação de canais, além da capacidade de adicionar e remover canais e sincronizar canais ativos.

Na subseção 3.3.1 foi detalhado o papel dos canais na comunicação entre o *host* e o dispositivo USB, os tipos de canais que podem ser criados e seus atributos e elenca dois canais pré-definidos de uso obrigatório do protocolo. Em seguida, na subseção 3.3.2 foi explicado o tratamento de prioridade de envio de uma mensagem. E por fim, na subseção 3.3.3 é determinado um conjunto de comandos para o funcionamento do PMCA.

3.3.1 Canais de comunicação

Os canais de comunicação são virtuais, ou seja, estão estabelecidos a nível lógico dentro de um único canal físico, que neste caso é o barramento USB. Eles são usados para direcionar o fluxo de dados entre dois dispositivos, onde cada canal pode ser configurado de maneira independente com diferentes prioridades de envio.

Existem dois tipos de canais: canal de configuração e canal de comunicação. O primeiro é responsável por definir as configurações e parâmetros da comunicação de outros canais que forem criados. É exclusivamente nesse canal que o módulo de controle de canais se concentra ao aplicar uma nova configuração, o qual deve ser único. Portanto, qualquer novo canal criado não pode ser do tipo configuração, ficando restrito ao tipo comunicação. Por sua vez, um canal de comunicação é utilizado para a transmissão regular de mensagens, ou seja, mensagens que não carregam nenhum comando para o módulo de controle de canais. Neste protocolo, é necessário que no mínimo dois canais estejam disponíveis durante o processo de comunicação. O que acarreta na necessidade de que ambos os canais sejam negociados entre as duas extremidades do protocolo. Os canais pré-definidos e de negociação obrigatória são: um canal de configuração e um canal de comunicação padrão. Os detalhes e características de cada canal podem ser consultados na Tabela 4.

Tabela 4 – Canais de comunicação padrão.

Identificador do canal	Tipo	Descrição
0	Configuração	Único canal utilizado para mensagens de configuração. Como pedidos de conexão, desconexão e sincronização de canais. Para outros pedidos veja a subseção 3.3.3
1	Comunicação	Canal padrão para troca de mensagens entre dois dispositivos.

Fonte: Elaborada pelo autor.

Cada canal criado possui um identificador único, o qual deverá ser indicado no

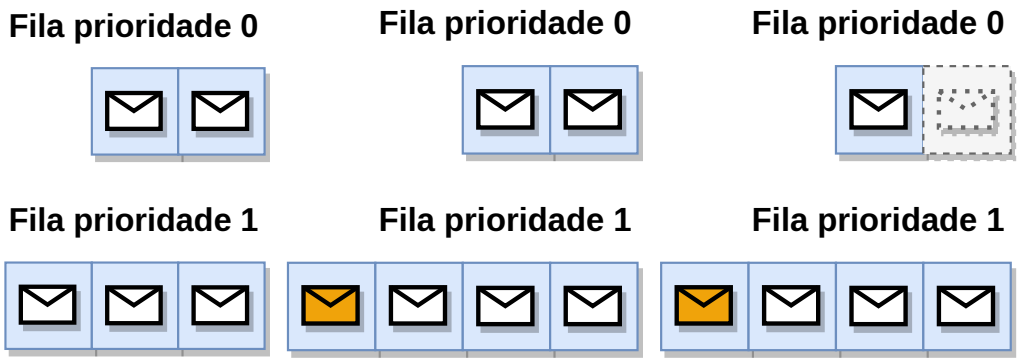
cabeçalho da mensagem, conforme descrito nas seções subseção 3.3.3 e subseção 3.3.4. Esse identificador, é um número de 0 a 255, no entanto, os valores zero e um já estão reservados para os canais padrões descritos acima. Desta forma, restam 254 canais que podem ser criados.

3.3.2 Expedição de mensagens

O controlador de canais realiza o envio das mensagens com base na prioridade atribuída a cada canal. A prioridade é representada por um número entre 1 e 255 para os canais de comunicação, e zero para o canal de configuração. A ordem de execução das mensagens depende exclusivamente da prioridade atribuída ao canal que a transporta. Quanto menor o valor da prioridade, maior é sua importância. Logo, o valor zero representa a maior prioridade, enquanto o valor 255 indica a prioridade mais baixa de envio.

Quando o módulo de controle de canais recebe uma nova mensagem, ele verifica a prioridade do canal associado a essa mensagem e a adiciona à fila correspondente. Por exemplo, considerando duas filas de envio, uma com prioridade zero e outra com prioridade um, a chegada de uma nova mensagem com prioridade um fará com que ela seja colocada no final da fila de prioridade um. Em seguida, seguindo a ordem de prioridades, se houver uma mensagem na fila de prioridade zero, a próxima mensagem a ser enviada será a primeira mensagem da fila de prioridade zero. Esse fluxo é exemplificado na Figura 8, onde a Figura 8a mostra as duas filas de envio, a Figura 8b apresenta a chegada de uma nova mensagem com prioridade 1 e a Figura 8c ilustra o envio subsequente da mensagem com prioridade zero.

Figura 8 – Fila de expedição de mensagens.



(a) Fila com prioridade 0 e fila com prioridade 1.

(b) Chegada de uma nova mensagem com prioridade igual a 1.

(c) Mensagem da fila com prioridade igual a 0 é enviada.

Fonte: Elaborada pelo autor.

3.3.3 Formatos das mensagens para o gerenciamento de canais

O estabelecimento e a manutenção dos canais de comunicação são regidos por um conjunto de mensagens específico definido para este módulo. Seu objetivo é estabelecer o significado dos dados trocados e determinar as ações a serem tomadas com base nesses dados. Tais comandos são apresentados na Tabela 5, fornecendo uma referência dos termos e comandos utilizados.

Tabela 5 – Conjunto de mensagens utilizado pelo módulo de Controle de Canais para a manutenção da conexão de canais.

Comando	Valor	Tipo	Caso de Uso	Descrição
`conectar`	`0x01`	Pedido	subseção 3.4.2	Conexão de um canal específico.
`confirma_conectar`	`0x02`	Resposta	subseção 3.4.2	Confirmação de conexão de um canal específico.
`nega_conectar`	`0x03`	Resposta	subseção 3.4.2	Conexão negada para um canal específico.
`desconectar`	`0x04`	Pedido	subseção 3.4.3	Desconexão de um canal específico.
`confirma_desconectar`	`0x05`	Resposta	subseção 3.4.3	Confirmação de desconexão de um canal específico.
`nega_desconectar`	`0x06`	Resposta	subseção 3.4.3	Desconexão negada para um canal específico.
`sincronizar`	`0x07`	Pedido	subseção 3.4.4	Sincronizar todos os canais ativos.
`confirma_sincronizar`	`0x08`	Resposta	subseção 3.4.4	Confirmação de sincronização de todos os canais ativos.
`restaurar`	`0x09`	Pedido	subseção 3.4.5	Desconexão de todos os canais ativos.
`confirma_restaurar`	`0x0A`	Resposta	subseção 3.4.5	Confirmação de desconexão de todos os canais ativos.
`erro`	`0xFA`	Resposta	subseção 3.4.2 e subseção 3.4.3	Mensagem de erro. Ao usar esse vocabulário, o byte seguinte deve possuir um código de erro presente na Tabela 10

Fonte: Elaborada pelo autor.

É importante ressaltar que todas as comunicações relacionadas à manutenção dos enlaces virtuais devem ser enviadas exclusivamente através do canal com identificador igual a `0` (zero). Essa designação assegura que as informações relevantes para a manutenção e configuração dos enlaces sejam direcionadas corretamente, evitando conflitos ou ambiguidades com outros canais de comunicação.

3.3.3.1 `conectar`

Este comando é um pedido que será enviado pelo dispositivo USB do tipo Mestre para o dispositivo Trabalhador para iniciar o processo de conexão de um canal específico entre os dispositivos. A Tabela 6 define o formato do quadro que deve ser enviado.

Ao receber este comando, o dispositivo Trabalhador irá verificar se o parâmetro `id_novo_canal` é válido, ou seja, se já não pertence a um canal existente, ou se ainda não atingiu o número máximo de canais. Em caso de sucesso, o dispositivo Trabalhador deve

Tabela 6 – Formato do campo do controlador de canais para o envio de comandos de conexão.

Nome do campo	Tamanho (bytes)	Valor	Descrição
`id_canal`	1	`0x00`	Número identificador do canal de configuração.
`comando`	1	variável	Comando de gerenciamento de canal (veja Tabela 5).
`id_canal_alvo`	1	variável	Número identificador do canal que será alvo da configuração.
`prioridade_novo_canal`	1	variável	Prioridade de acesso ao barramento do canal.
`tamanho`	variável	variável	Tamanho máximo que a mensagem pode ter. O valor `0` (zero) deve ser definido caso não haja limite.

Fonte: Elaborada pelo autor.

gerar uma resposta com o comando `confirma_conectar`. Em caso de insucesso, será o comando `nega_conectar` que deverá ser enviado.

3.3.3.2 `confirma_conectar`

Este comando só deve ser enviado caso o Trabalhador tenha recebido previamente um comando `conectar`. O `confirma_conectar` é uma resposta que será enviada pelo dispositivo USB do tipo Trabalhador para o dispositivo Mestre para confirmar a conexão de um canal específico entre os dispositivos. A Tabela 6 define o formato do quadro que deve ser enviado.

3.3.3.3 `nega_conectar`

Este comando só deve ser enviado caso o Trabalhador tenha recebido previamente um comando `conectar`. O `nega_conectar` é uma resposta que será enviada pelo dispositivo USB do tipo Trabalhador para o dispositivo Mestre para negar a conexão de um canal específico entre os dispositivos. A Tabela 6 define o formato do quadro que deve ser enviado.

Em seguida, o Trabalhador deverá também enviar outra mensagem com o comando `erro` justificando a razão de haver negado a conexão do canal.

3.3.3.4 `desconectar`

Este comando é um pedido que será enviado pelo dispositivo USB do tipo Mestre para o dispositivo Trabalhador para iniciar o processo de desconexão de um canal específico entre os dispositivos. A Tabela 6 define o formato do quadro que deve ser enviado.

Ao receber este comando, o dispositivo Trabalhador irá verificar se o parâmetro `id_novo_canal` é válido, ou seja, se pertence a um canal existente ou se os identificadores dos canais são diferentes zero e um, pois tais canais não podem ser desconectados. Em caso de sucesso, o dispositivo Trabalhador deve gerar uma resposta com o comando `confirma_desconectar`. Em caso de insucesso, será o comando `nega_desconectar` que deverá ser enviado.

3.3.3.5 `confirma_desconectar`

Este comando só deve ser enviado caso o Trabalhador tenha recebido previamente um comando `desconectar`. O `confirma_desconectar` é uma resposta que será enviada pelo dispositivo USB do tipo Trabalhador para o dispositivo Mestre para confirmar a desconexão de um canal específico entre os dispositivos. A Tabela 6 define o formato do quadro que deve ser enviado.

3.3.3.6 `nega_desconectar`

Este comando só deve ser enviado caso o Trabalhador tenha recebido previamente um comando `desconectar`. O `nega_desconectar` é uma resposta que será enviada pelo dispositivo USB do tipo Trabalhador para o dispositivo Mestre para negar a desconexão de um canal específico entre os dispositivos. A Tabela 6 define o formato do quadro que deve ser enviado.

Em seguida, o Trabalhador deverá também enviar outra mensagem com o comando `erro` justificando a razão de haver negado a desconexão do canal.

3.3.3.7 `sincronizar`

Este comando é um pedido que será enviado pelo dispositivo USB do tipo Mestre para o dispositivo Trabalhador para iniciar o processo de sincronização de todos os canais ativos no Trabalhador. A Tabela 7 define o formato do quadro que deve ser enviado.

Tabela 7 – Formato do campo do controlador de canais para o envio do comando de pedido de sincronização e restauração.

Nome do campo	Tamanho (bytes)	Valor	Descrição
`id_canal`	1	`0x00`	Número identificador do canal de configuração.
`comando`	1	variável	Comando de gerenciamento de canal (veja Tabela 5).

Fonte: Elaborada pelo autor.

Ao receber este comando, o dispositivo Trabalhador deve gerar uma resposta com o comando `confirma_sincronizar`.

3.3.3.8 `confirma_sincronizar`

Este comando só deve ser enviado caso o Trabalhador tenha recebido previamente um comando `sincronizar`. O `confirma_sincronizar` é uma resposta que será enviada pelo dispositivo USB do tipo Trabalhador para o dispositivo Mestre para confirmar a sincronização de todos os canais entre os dispositivos. A Tabela 8 define o formato do quadro que deve ser enviado.

Tabela 8 – Formato do campo do controlador de canais para o envio do comando de resposta de sincronização.

Nome do campo	Tamanho (bytes)	Valor	Descrição
`id_canal`	1	`0x00`	Número identificador do canal de configuração.
`comando`	1	variável	Comando de gerenciamento de canal (veja Tabela 5).
`nu_canais`	1	variável	Quantidade de canais para serem sincronizados.
`id_canais`	`nu_canais`	variável	Identificadores dos canais a serem sincronizados.
`pr_canais`	`nu_canais`	variável	Prioridade de envio dos canais a serem sincronizados.

Fonte: Elaborada pelo autor.

Ao associar os campos `id_canais` e `pr_canais` a listas, estabelecemos que o mesmo índice em cada uma delas está relacionado a um mesmo canal. Em outras palavras, o segundo *byte* do campo `id_canais` e o segundo *byte* do campo `pr_canais` pertencem ao mesmo canal, por exemplo.

3.3.3.9 `restaurar`

Este comando é um pedido que será enviado pelo dispositivo USB do tipo Mestre para o dispositivo Trabalhador para iniciar o processo de restauração de todos os canais ativos no Trabalhador. A Tabela 7 define o formato do quadro que deve ser enviado.

Ao receber este comando, o dispositivo Trabalhador deve gerar uma resposta com o comando `confirma_restaurar`.

3.3.3.10 `confirma_restaurar`

Este comando só deve ser enviado caso o Trabalhador tenha recebido previamente um comando `restaurar`. O `confirma_restaurar` é uma resposta que será enviada pelo dispositivo USB do tipo Trabalhador para o dispositivo Mestre para confirmar a restauração de todos os canais entre os dispositivos. A Tabela 7 define o formato do quadro que deve ser enviado.

3.3.3.11 `erro`

Este comando só deve ser enviado caso o Trabalhador tenha enviado previamente um comando `nega_conectar` ou `nega_desconectar`. O `erro` é uma resposta que será enviada pelo dispositivo USB do tipo Trabalhador para o dispositivo Mestre para justificar o erro causado pela recepção de um pedido. A Tabela 9 define o formato do quadro que deve ser enviado. Por sua vez, a Tabela 10 elenca os possíveis erros que podem ser retornados neste comando.

Tabela 9 – Formato do campo do controlador de canais para o envio do comando de erro.

Nome do campo	Tamanho (bytes)	Valor	Descrição
`id_canal`	1	`0x00`	Número identificador do canal de configuração.
`erro`	1	variável	Valor que justifica o erro (veja Tabela 10).

Fonte: Elaborada pelo autor.

Tabela 10 – Mensagens de erros da manutenção de canais.

Vocabulário	Valor	Descrição
`desconhecido`	`0x00`	Erro não identificado.
`max_canais`	`0x01`	Número máximo de canais atingido.
`id_invalido`	`0x02`	Identificador do canal inválido.
`cmd_invalido`	`0x03`	Comando inválido.

Fonte: Elaborada pelo autor.

3.3.4 Formato das mensagens para o envio de dados regulares

Na Tabela 11 é apresentado o formato do quadro do PMCA, o qual deve ser utilizado em todo envio de mensagens regulares, ou seja, mensagens que não possuem um comando de configuração do protocolo.

Tabela 11 – Formato do campo do controlador de canais para o envio do comando de mensagens regulares.

Nome do campo	Tamanho (bytes)	Valor	Descrição
`id_canal`	1	`0x00`	Número identificador do canal de comunicação.
`conteúdo`	1	variável	Dados transmitidos.

Fonte: Elaborada pelo autor.

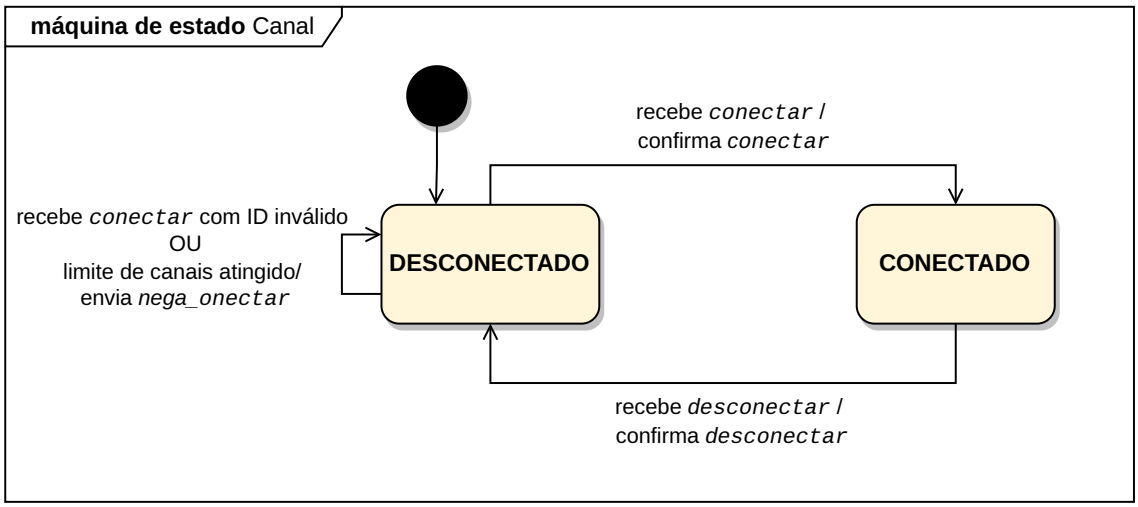
3.3.5 Gerenciamento de conexão entre canais

O estabelecimento de um canal de comunicação segue o modelo pare-e-espere de forma que, para cada comando trocado entre Mestre e Trabalhador deve haver uma con-

firmação de entrega. Esse processo de confirmação assegura que os comandos sejam transmitidos de forma correta e que ambos os lados estejam sincronizados durante a troca de informações.

Inicialmente, assim que um canal é criado pelo Mestre, o canal é configurado no estado `DESCONECTADO`. Na outra extremidade do protocolo, quando o comando `conectar` é recebido pelo Trabalhador, ele emite uma resposta de confirmação (`confirma_conectar`) e transita para o estado `CONECTADO`. O Trabalhador, ao receber essa confirmação, também altera o estado para `CONECTADO` ao canal correspondente. Essa sequência de estados e transições permite a sincronização das extremidades e estabelece a conexão necessária para a troca de informações entre as partes envolvidas. Para uma melhor compreensão, a Figura 9 apresenta uma representação visual desse processo de estabelecimento do canal de comunicação.

Figura 9 – Máquina de estados finita que representa a troca de estado de um Canal.



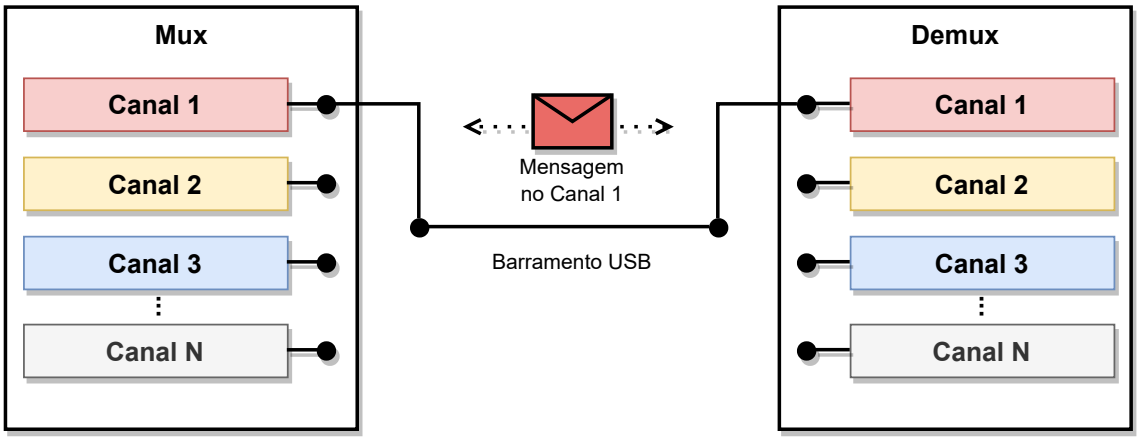
Fonte: Elaborada pelo autor.

Uma vez que um canal é estabelecido, o controlador de canais cria enlaces virtuais, em que o remetente e o destinatário do canal possuem o mesmo identificador. Esses enlaces virtuais garantem que a comunicação seja direcionada corretamente. A Figura 10 ilustra esse processo, demonstrando como o controlador de canais cria os enlaces virtuais para cada canal estabelecido.

3.4 Descrição dos casos de uso

Esta seção apresenta uma série de casos de uso que foram desenvolvidos para ilustrar e demonstrar as diferentes situações em que o protocolo é aplicado. Cada cenário tem suas interações descritas passo a passo entre os diferentes componentes e entidades envolvidas.

Figura 10 – Comutação entre canais.



Fonte: Elaborada pelo autor.

3.4.1 Inicialização do protocolo

- **Resumo:** Este caso de uso descreve as etapas para a inicialização do protocolo para que ele esteja apto a enviar de mensagens.
- **Ator primário:** Mestre
- **Pré-condições:** Dispositivo Mestre e Trabalhador devem estar conectados a uma porta USB em um mesmo *host*.
- **Fluxo principal:**
  - O **Mestre** envia o comando `conectar` com o ID do canal igual a zero o Trabalhador.
  - O **Trabalhador** recebe o comando `conectar` e responde com o comando `confirma_conectar` com o ID do canal igual a zero.
  - O **Mestre** envia o comando `conectar` com o ID do canal igual a um o Trabalhador.
  - O **Trabalhador** recebe o comando `conectar` e responde com o comando `confirma_conectar` com o ID do canal igual a um.

3.4.2 Nova Conexão

- **Resumo:** Este caso de uso descreve as etapas para a criação de um novo canal de comunicação no protocolo.
- **Ator primário:** Mestre
- **Pré-condições:** Canal 0 deve estar no estado `CONECTADO`.

- **Fluxo principal:**
  - O **Mestre** cria um canal com ID igual a N e atribui o estado `DESCONECTADO` a ele.
  - O **Mestre** envia o comando `conectar` com o ID do novo canal igual a N para o Trabalhador.
  - O **Trabalhador** recebe o comando `conectar` com o ID do novo canal igual a N, cria esse canal e atribui o estado `CONECTADO` a ele.
  - O **Trabalhador** responde com o comando `confirma_conectar` com o ID do novo canal igual a N.
  - O **Mestre** recebe o comando `confirma_conectar` e atualiza o estado de conexão do canal N para `CONECTADO`.
- **Fluxo Alterativo:** Limite de canais atingido ou ID inválido.
  - O **Trabalhador** não cria nenhum canal e responde com o comando `nega_conectar` com o ID do novo canal igual a N.
  - O **Trabalhador** responde com o comando `erro` com um valor da Tabela 10 que justifique o motivo de ter recusado o pedido de conexão.
  - O **Mestre** recebe os comandos e desiste da conexão.

3.4.3 Encerrar Conexão

- **Resumo:** Este caso de uso descreve as etapas para a exclusão de um canal de comunicação existente no protocolo.
- **Ator primário:** Mestre
- **Pré-condições:** Canal 0 deve estar no estado `CONECTADO`.
- **Fluxo principal:**
  - O **Mestre** envia o comando `desconectar` com o ID do canal a ser excluído igual a N para o Trabalhador.
  - O **Trabalhador** recebe o comando `desconectar` com o ID do canal a ser excluído igual a N e atribui o estado `DESCONECTADO` a ele.
  - O **Trabalhador** responde com o comando `confirma_desconectar` com o ID do canal a ser excluído igual a N.
  - O **Mestre** recebe o comando `confirma_desconectar` e atualiza o estado de conexão do canal N para `DESCONECTADO`.

- **Fluxo Alterativo:** ID inválido.
  - O **Trabalhador** responde com o comando `nega_desconectar` com o ID do canal a ser excluído igual a N.
  - O **Trabalhador** responde com o comando `erro` com um valor da Tabela 10 que justifique o motivo de ter recusado o pedido de desconexão.
  - O **Mestre** recebe os comandos e desiste da desconexão.

3.4.4 Sincronização de canais

- **Resumo:** Este caso de uso descreve as etapas para a sincronização de canais ativos.
- **Ator primário:** Mestre
- **Pré-condições:** Canal 0 deve estar no estado `CONECTADO`.
- **Fluxo principal:**
  - O **Mestre** envia o comando `sincronizar`.
  - O **Trabalhador** recebe o comando `sincronizar` e responde com o comando `confirma_sincronizar` contendo todos os canais ativos no Trabalhador.
  - O **Mestre** recebe o comando `confirma_sincronizar` e para cada canal recebido, atualiza o seu estado para `CONECTADO`.

3.4.5 Restauração de canais

- **Resumo:** Este caso de uso descreve as etapas para a restauração de canais ativos, fazendo com que todos os canais previamente criados pelo usuário sejam excluídos.
- **Ator primário:** Mestre
- **Pré-condições:** Canal 0 deve estar no estado `CONECTADO`.
- **Fluxo principal:**
  - O **Mestre** envia o comando `restaurar`.
  - O **Trabalhador** recebe o comando `restaurar`, deleta todos os canais ativos com identificador diferente de zero ou um responde com o comando `confirma_restaurar`.
  - O **Mestre** recebe o comando `confirma_restaurar` e deleta todos os canais ativos com identificador diferente de zero ou um.

3.4.6 Envio de Mensagem

- **Resumo:** Este caso de uso descreve as etapas para o envio de mensagens.
- **Ator primário:** Mestre
- **Pré-condições:** Um canal com identificador superior a zero deve estar no estado `CONECTADO`.
- **Fluxo principal:**
  - O **Mestre** envia uma mensagem através de um Canal N, para o Trabalhador.
  - O **Trabalhador** recebe a mensagem.

3.5 Considerações finais

Em conclusão, este capítulo apresentou as definições e regras necessárias para a implementação do protocolo proposto, que oferece uma variedade de serviços essenciais para a comunicação. Com base nessas definições, o Capítulo 4 irá abordar a execução do protocolo em um cenário de testes. Será possível avaliar o uso do protocolo em ação, observando como ele lida com o gerenciamento de canais, o controle de prioridades e outros aspectos relevantes. Essa fase de testes permitirá verificar se o protocolo atende às necessidades específicas e se cumpre as propostas aqui definidas.

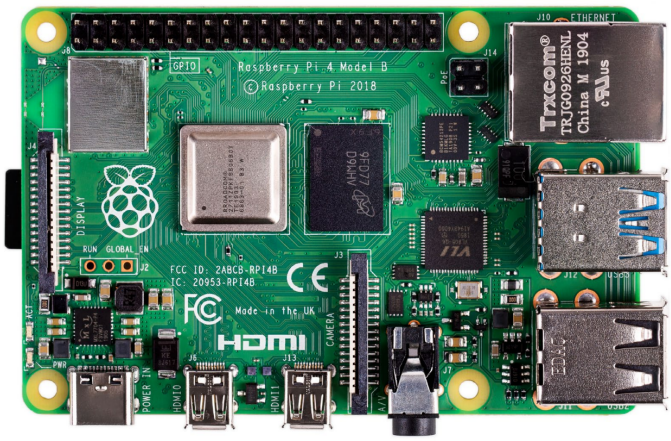
4 EXPERIMENTOS E VALIDAÇÃO

A fim de apurar o funcionamento do protocolo proposto no Capítulo 3, foi criada uma aplicação fictícia que consiste em um dispositivo responsável pela aquisição de dados. Esse dispositivo coleta informações de temperatura e umidade e as envia periodicamente para o *host*. Além disso, possui a capacidade de enviar arquivos de texto e imagens, bem como receber comandos específicos da aplicação por meio de um canal dedicado. A fim de verificar a confiabilidade da implementação, criaram-se testes que verificam a capacidade do PMCA em lidar com situações adversas, como interrupções de conexão, erros de transmissão e congestionamento de rede. Foram realizados testes de transferência de dados em condições controladas e testes de estresse para avaliar a robustez do protocolo.

Para a implementação dos módulos e a transferência de dados entre dispositivos, foi escrito um programa usando a linguagem Python3[1]. A biblioteca pySerial[2] foi empregada para facilitar a comunicação entre os dispositivos.

Existem dois atores neste experimento: o dispositivo USB e o *host*. O primeiro é uma *Raspberry Pi 4 Model B*, apresentada na Figura 11, a qual está executando o sistema operacional *Raspberry Pi OS*. A utilização desse modelo de Raspberry se dá em função de que ela possui o recurso de USB-On-The-Go (OTG), o qual é necessário para poder configurá-la como um *gadget* e por consequência como um dispositivo do tipo USB-CDC. Por sua vez, o *host* é um computador que está executando o sistema operacional *Linux*, que possui nativamente o *driver* que realiza a comunicação via USB-CDC.

Figura 11 – Uma Raspberry Pi 4 Model B.



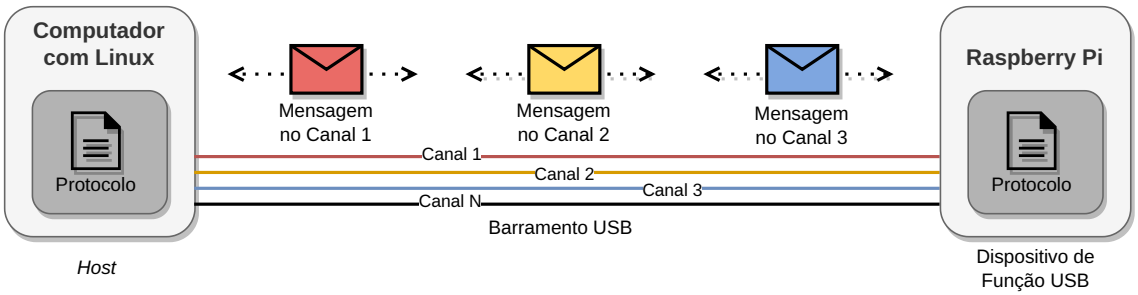
Fonte: (RASPBERRY, 2021)

Após realizar a inicialização do *script* em *Python* que contém a aplicação fictícia

[1] https://www.python.org/
[2] https://github.com/pyserial

que faz uso do protocolo em ambas as extremidades, com um cabo USB foi estabelecida a conexão entre ambos dispositivos conforme exibe a Figura 12. Imediatamente, o *host* pôde ser capaz de identificar essa conexão e automaticamente foram estabelecidos os dois canais de comunicação padrão descritos na subseção 3.3.1. Em seguida, através da aplicação do *host* executou-se o pedido de criação de um novo canal de comunicação, o qual foi adicionado corretamente, totalizando assim dois canais de comunicação e um canal de configuração ativos no protocolo.

Figura 12 – Cenário de execução dos testes.



Fonte: Elaborada pelo autor.

Através desses canais foram realizados alguns testes de envios. Inicialmente o *host* enviou mensagens de texto simples, as quais foram exibidas pela aplicação do dispositivo USB. Em seguida, o *host* enviou arquivos de texto e imagens pelos canais, os quais chegaram íntegros na outra extremidade do protocolo. Com o intuito de verificar a fila de prioridades de envio, foi iniciado um teste onde por meio do Canal 1 eram enviados periodicamente pelo *host* um pedido de coleta de dados de temperatura e umidade. No Canal 2, por sua vez, eram enviadas periodicamente arquivos de texto. E por último, no Canal 3, um usuário inseria manualmente comandos via terminal que deveriam ser executados o mais breve possível pelo dispositivo USB. Desta forma, o canal e as suas respectivas prioridades foram definidas conforme a Tabela 12.

Tabela 12 – Canais do cenário de testes.

Identificador do canal	Prioridade	Descrição
0	0	Canal de configuração padrão.
1	2	Canal de comunicação padrão que é responsável por tratar os pedidos de coleta de dados.
2	2	Canal de comunicação que é responsável por tratar envio de arquivos de texto.
3	1	Canal de comunicação que é responsável por tratar comandos enviados pelo usuário em tempo real.

Fonte: Elaborada pelo autor.

Durante os testes, verificou-se que as mensagens foram corretamente adicionadas às suas respectivas filas de envio, possibilitando uma execução adequada da comunicação.

Foi realizado o teste de interrupção de conexão a partir da remoção do cabo USB que conectava ambos os dispositivos. Ao realizar esse procedimento, o *host* abortou todas as suas transmissões, pois detectou que o dispositivo foi removido. Em seguida, no *host* foi forçada a exclusão de todos os canais ativos, e logo que o cabo USB foi reconectado, o *host* identificou essa conexão e enviou um pedido de sincronização de canais para o dispositivo USB, o qual respondeu corretamente e todos os canais ativos no dispositivo USB foram novamente configurados no *host*.

Em um outro cenário, foi executado um teste de taxa de transmissão, onde realizou-se a ativação de cinco canais simultâneos para avaliar o desempenho da rede. A quantidade de *bytes* recebidos em cada canal foi calculada levando em consideração o momento de envio e até a chegada dos dados, o que possibilitou uma análise da velocidade de transmissão. Todos esses dados foram compilados e apresentados na Tabela 13.

Tabela 13 – Medições realizadas para testes de taxa transmissão de dados.

Canais	**Tamanho (bytes)**	**Medição 1 (Mbps)**	**Medição 2 (Mbps)**	**Medição 3 (Mbps)**	**Medição 4 (Mbps)**	**Medição 5 (Mbps)**	**Média (Mbps)**
5	100	4.001	4.159	3.999	3.913	3.998	4,014
5	500	3.731	4.159	3.602	3.913	3.997	3.880
5	1000	3.944	3.595	3.794	3.731	4.191	3.861

Fonte: Elaborada pelo autor.

Além da análise da taxa de transmissão, também foi realizada uma avaliação da taxa de erro durante o teste. Neste o processo, uma quantidade específica de *bytes* foi enviada através dos cinco canais ativos, e, em seguida, foi verificado se a mesma quantidade de *bytes* foi recebida. Esse procedimento permitiu identificar qualquer perda de dados ao longo da transmissão. Além disso, os dados recebidos foram comparados com os dados enviados para garantir a integridade dos dados, verificando se não ocorreram corrupções ou alterações indesejadas durante o trajeto. Todos esses dados estão dispostos na Tabela 14.

Tabela 14 – Medições realizadas para testes de taxa de erro.

Canais	**Tamanho (bytes)**	**Medição 1 (bytes)**	**Medição 2 (bytes)**	**Medição 3 (bytes)**	**Medição 4 (bytes)**	**Medição 5 (bytes)**	**Sucesso**
5	100	100	100	100	100	100	100%
5	1000	1000	1000	1000	1000	1000	100%
5	100000	100000	100000	100000	100000	100000	100%

Fonte: Elaborada pelo autor.

Após a execução de todos esses cenários, foi concluído que os canais de comunicação são configurados corretamente e mudam de estado conforme as ações do Mestre e do Trabalhador. A troca de informações ocorre de forma adequada em cada canal respeitando a fila de prioridades. As conexões e desconexões são executadas corretamente,

interrompendo ou estabelecendo a comunicação conforme o esperado. O protocolo mantém o desempenho esperado mesmo em condições de estresse.

5 CONCLUSÃO

Este trabalho buscou oferecer uma solução de comunicação de múltiplos canais assíncronos através do protocolo USB, de forma que a sua implementação eliminasse a dependência de *drivers* específicos, facilitando assim a integração de dispositivos de diferentes fabricantes.

Para alcançar tais objetivos, foram realizadas pesquisas e análises das tecnologias existentes, com o intuito de identificar as melhores práticas e abordagens a serem adotadas. Esses fundamentos foram essenciais para a proposição do PMCA.

Foram elencados alguns serviços providos pelo protocolo proposto sendo estes a multiplexação dos dados, que permite a transmissão simultânea de múltiplas fontes de dados através de um único canal de comunicação; o controle de prioridades, pois ele permite gerenciar a ordem de prioridade das mensagens transmitidas, garantindo que mensagens críticas ou urgentes sejam entregues prontamente, mesmo em situações de alto tráfego. Além disso, o protocolo inclui um processo de *handshaking* para estabelecer uma conexão entre os dispositivos de comunicação. Esse processo envolve uma troca de sinais entre o transmissor e o receptor para verificar a disponibilidade e a capacidade de comunicação. Outro serviço oferecido pelo protocolo é a negociação de parâmetros de comunicação. Isso permite que os dispositivos comuniquem suas capacidades e requisitos específicos durante o processo de estabelecimento da conexão. Por fim, com a negociação de parâmetros, os dispositivos podem ajustar configurações ideais para a comunicação, como formato dos dados, controle de erros, entre outros, com base em suas capacidades e necessidades.

Uma implementação da solução foi realizada com base no modelo proposto. A qual foi feita utilizando uma *Raspberry Pi 4 Model B* e um computador com sistema operacional *Linux*, utilizando a biblioteca *pySerial* em *Python*. Posteriormente, foram conduzidos experimentos para validar o protocolo proposto. Nos cenários de teste, o protocolo demonstrou um comportamento conforme o esperado, mesmo em situações adversas, como interrupção drástica da comunicação e sobrecarga no envio de dados. Em todos os cenários, o protocolo foi capaz de se recuperar, quando necessário, e de manter e atualizar corretamente os estados de cada módulo.

Os testes realizados comprovaram que a proposta apresentada neste trabalho possibilita a troca de dados entre um *host* e um dispositivo USB. Além disso, os testes também validaram a implementação para o cenário específico abordado. Apesar dos resultados positivos obtidos, é importante reconhecer que ainda há espaço para futuras pesquisas e aprimoramentos. O protocolo proposto pode ser estendido para outros cená-

rios e casos de uso, considerando diferentes variáveis e contextos específicos. Além disso, é fundamental continuar testando e refinando o protocolo, garantindo sua compatibilidade e desempenho em diferentes ambientes.

5.1 Trabalhos futuros

A fim de verificar todas as propriedades do protocolo, uma simples observação humana, por meio de uma máquina de estados finita, pode não garantir que as propriedades do protocolo não sejam violadas e de que eventos como *deadlocks*, *livelocks* ou encerramentos inesperados ocorram. Nesse caso, faz-se necessário uma validação de suas propriedades com o propósito de verificar as suas funcionalidades e robustez. Uma opção é fazer uso de uma linguagem para modelagem de sincronização e coordenação de processos concorrentes PROMELA e o verificador SPIN. O PROMELA é capaz de descrever protocolos de comunicação por meio da criação de canais de mensagens, o modelo permite a análise de mensagens síncronas ou assíncronas. Um benefício dessa linguagem é a sua abstração aos detalhes do modelo que não são relacionados à interação entre os processos que modelam o comportamento dos canais. Já o SPIN irá realizar a verificação do que será implementado em PROMELA e validar o modelo.

Este trabalho realizou a implementação do protocolo seguindo o modelo mestre e trabalhador. No entanto, visando ampliar os cenários de aplicação, é possível realizar uma alteração que permita que ambas as extremidades do protocolo tenham a capacidade de ter a iniciativa de enviar mensagens. Embora o protocolo USB possua métodos de controle de acesso ao meio, é necessário realizar um estudo para determinar se é necessário implementar uma camada adicional de controle de acesso ao meio para evitar colisões durante o envio de dados entre os dispositivos envolvidos.

Uma possível adição ao protocolo é a inclusão de um módulo fragmentador de pacotes. Esse módulo seria útil quando o tamanho máximo do quadro enviado precisar ser limitado em nível de protocolo de comunicação, e não apenas na aplicação. Ele seria especialmente apropriado para dispositivos com recursos de memória limitados. O módulo fragmentador seria responsável por dividir um pacote em blocos menores durante o envio e reagrupar todos os blocos de uma mensagem na recepção. Para isso, seria necessário adicionar um campo no cabeçalho da mensagem para identificar o número de sequência da mensagem e uma forma de determinar quando o último bloco da mensagem foi entregue.

Se houver a necessidade de utilizar o protocolo em um ambiente em que a entrega das mensagens precisa ser garantida, pode ser necessário adicionar um módulo de garantia de entrega. Esse módulo estabeleceria métodos de confirmação de entrega e permitiria retransmissões de mensagens, garantindo assim a confiabilidade na transmissão. Embora o protocolo USB forneça esse tipo de proteção, é importante analisar cada cenário in-

dividualmente para determinar sua aplicabilidade. Por exemplo, pode ser uma útil ter um mecanismo que permita ao remetente colocar o receptor em um estado conhecido, caso algum *software* anterior tenha enviado dados inválidos e esse tratamento pode ser realizado através das confirmações das garantias de entrega.

Essas adições ao protocolo podem expandir sua flexibilidade e capacidade de lidar com diferentes requisitos de comunicação. No entanto, é importante avaliar cuidadosamente a necessidade de cada módulo adicional, considerando os recursos disponíveis nos dispositivos envolvidos e as exigências específicas do ambiente de aplicação. Também deve ser levado em conta os benefícios e possíveis impactos na eficiência e complexidade do protocolo.

REFERÊNCIAS

APPLE et al. *Universal Serial Bus 3.2 Specification*. [s.n.], 2017. Disponível em: https://www.usb.org/document-library/usb-32-specification-released-september-22-2017-and-ecns. 29

AXELSON, J. *USB Complete: The Developer's Guide*. [S.l.]: Lakeview Research, 2015. 27, 28, 29, 32

CHRISTOFOROU, E. et al. Algorithmic mechanisms for reliable master-worker internet-based computing. *IEEE Transactions on Computers*, v. 63, n. 1, p. 179–195, 2014. 39

COMPAQ et al. *Universal Serial Bus Specification*. [s.n.], 2000. Universal Serial Bus Specification Revision 2.0. Disponível em: https://www.usb.org/document-library/usb-20-specification. 23, 24, 27, 28, 29, 30, 31, 32, 40

HOLZMANN, G. J. *Design And Validation Of Computer Protocols*. [S.l.]: Prentice Hall Software Series, 1991. 33, 34, 35, 36

KARNE, R. K. et al. A bare pc mass storage usb driver. *Computer and Information Sciences,Towson University*, 2012. 23

MICROCHIP (Ed.). *USB Speeds and Specifications*. 2020. Disponível em: https://microchipdeveloper.com/usb:speeds. 29, 31

RASPBERRY (Ed.). *Raspberry Pi 4 Model Product Brief*. 2021. Disponível em: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf. 57